# 课程回顾

- 标准记号及常用函数（单调性、取整、模运算、多项式、指数、对数、阶乘、多重函数、多重对数函数、斐波那契数）
- NP完全性理论
  - 问题内在复杂性“难”or“易”
  - 优化问题/判定问题
  - P/NP/NPC问题

# 归约 (reduce)

- 多项式时间归约算法 (reduction algorithm)，将A的任何实例α转化成B的某个实例β：
  - 转换操作需要多项式时间
  - 两个实例的解是相同的（α的解是“是”iff β的解也是“是”）

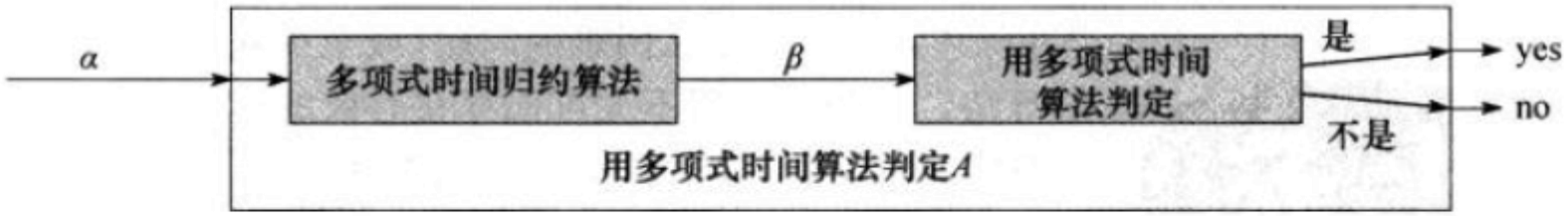



图 34-1 在给定另一个问题 $B$ 的多项式时间判定算法后，如何利用多项式时间归约算法在多项式时间内解决判定问题 $A$。将 $A$ 的实例 $\alpha$ 在多项式时间内转换为 $B$ 的实例 $\beta$，在多项式时间内解决 $B$，再将 $\beta$ 作为 $\alpha$ 的解

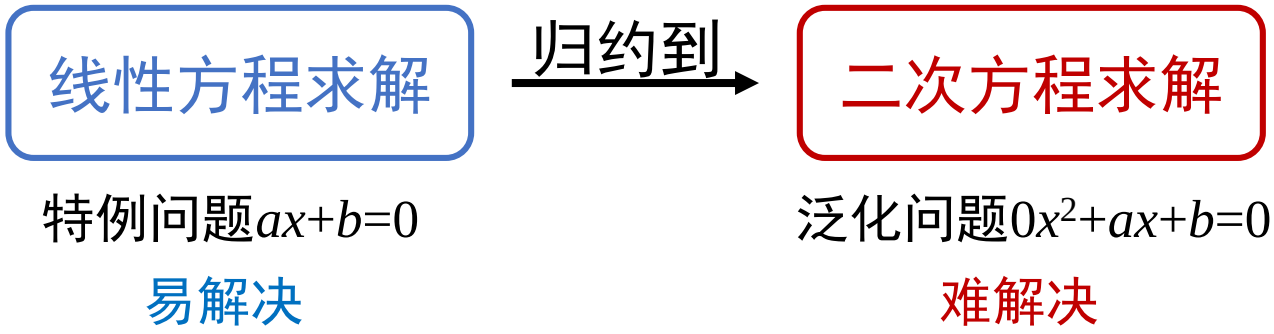
# 归约 (reduce，续)

- 将对问题A的求解“归约”到对问题B的求解，可利用问题B的“易求解性”来证明A的“易求解性”
- 相反的，假设有判定问题A，已知它不可能存在多项式时间算法，若有一个多项式时间归约，将A的一个实例转化为一个B的实例，则B也不可能存在多项式时间算法
- A的求解归约到B的求解（或者说问题A归约到问题B），则问题B难于问题A，或者说A并不比B更难解决
  - A的任何实例都可以被“容易地重新描述”为B的实例，B的实例的解也是A的实例的解

# 归约 (reduce，续)

■例：求解关于未知量$x$的线性方程问题可以转化为求解二次方程问题

➢实例$ax+b=0$可变换为$0x^2+ax+b=0$



# NPC类问题 (续)

■若对于问题Q，满足：

1. Q∈NP
2. 每个NP问题都可以多项式时间归约到Q

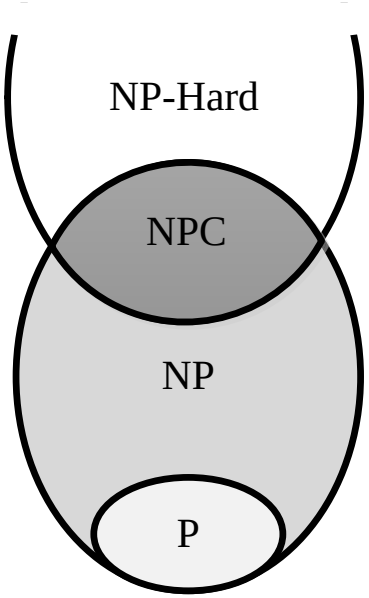
   或：任意一个NPC问题都可以多项式时间归约到Q

那么问题Q是一个NP完全问题

■若问题Q满足了第二条，则称为NP-Hard问题

# NP完全性理论

- 定理34.4 (p628) 如果任何NP完全问题是多项式时间可求解的，则P=NP。等价地，如果存在某一NP中的问题不是多项式时间可求解的，则所有NP完全问题都不是多项式时间可求解的。
- 目前认为P≠NP



# Karp的21个NP完全问题

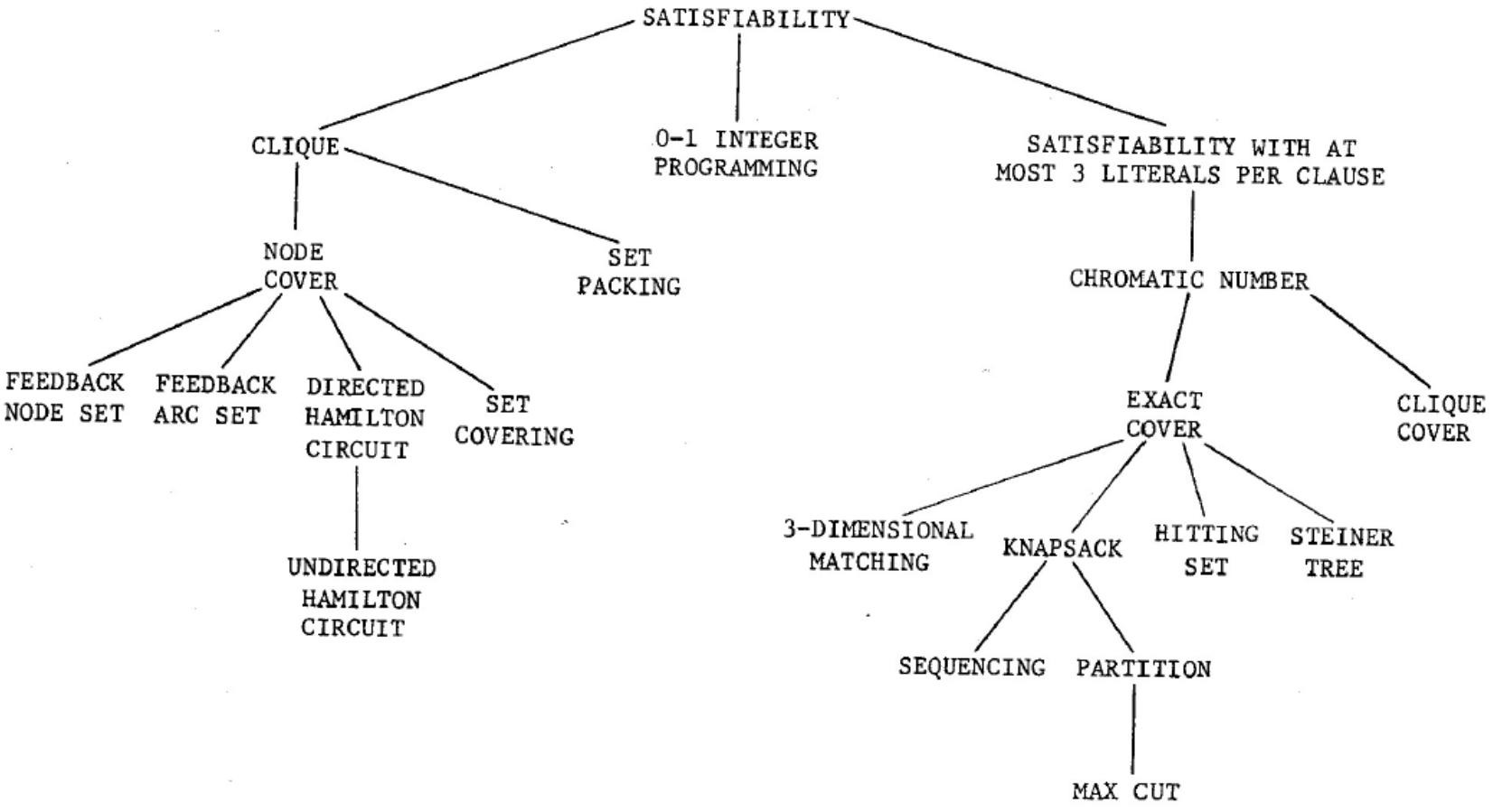



FIGURE 1 - Complete Problems

# 典型的NPC问题

■合取范式的可满足性问题$k$-CNF-SAT

➢布尔表达式 $(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_3) \wedge (\neg x_2 \vee \neg x_3)$

- 布尔变量$x_i$，取值为0或1
- 布尔连接词：例如∧（与，合取）、∨（或，析取）、¬（非）
- 括号

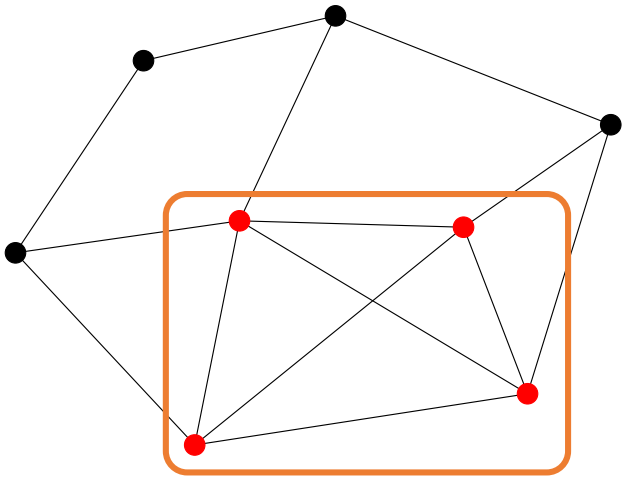
➢合取范式 (Conjunctive Normal Form, CNF)

- 布尔表达式用合取连接若干个析取子句
- 若每个子句恰好有$k$个布尔变量或其否定形式，则称为$k$合取范式（$k$-CNF）
- 2-CNF-SAT∈P，但3-CNF-SAT∈NPC $\Omega\left(2^n\right)$

# 典型的NPC问题 (续)

■团问题CLIQUE

- 给定一个无向图$G=(V, E)$和一个正整数$k$，判定图$G$是否包含一个$k$团，即是否存在$V' \subseteq V$和$|V'|=k$，且对任意$u,w \in V'$，有$(u,w) \in E$。
- 朴素算法：找出所有$k$个顶点的子图并验证



$$\Omega\left(C_{|V|}^{k} C_{k}^{2}\right)=\Omega\left(k^{2} C_{|V|}^{k}\right)$$

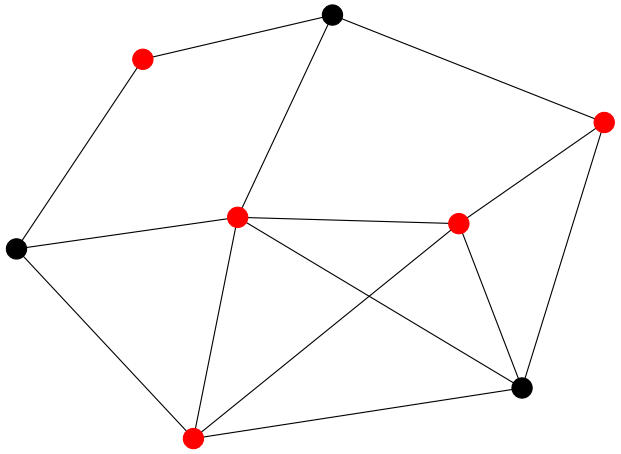
$k$为常数，则为$\Omega(|V|^k)$
$k$通常与$|V|$相关，则为超多项式级别

存在一个规模$k$为4的团

# 典型的NPC问题 (续)

■顶点覆盖问题VERTEX-COVER

- 给定一个无向图$G=(V, E)$和一个正整数$k$，判定是否存在$V' \subseteq V$和$|V'|=k$，使得对任意$(u,v) \in E$有$u \in V'$或$v \in V'$，如果存在，就称$V'$为图$G$的一个大小为$k$的顶点覆盖
- 即$E$中每条边关联的两个顶点至少有一个在$V'$中



$$\Omega\left(|E|C_{|V|}^{k}\right)$$

存在一个规模$k$为5的顶点覆盖

# 典型的NPC问题 (续)

■子集和问题SUBSET-SUM

- 给定一个正整数有限集$S$和一个整数目标$t>0$，判定是否存在$S$的一个子集$S' \subseteq S$，使得$S$’中的整数的和为$t$
- 例：$S$={1, 2, 7, 14, 49, 98, 343, 686, 2409, 2793, 16808, 17206, 117705, 117993}, $t$ =138457，则子集$S$’={1, 2, 7, 98, 343, 686, 2409, 17206, 117705}是该问题的一个解

$$\Omega\left(2^n\right)$$

# 典型的NPC问题 (续)

■哈密顿回路问题HAM-CYCLE

- 给定无向图$G=(V, E)$判定其是否含有一条哈密顿回路
- 无向图$G=(V, E)$中的一条哈密顿回路是通过$V$中每个顶点的简单回路

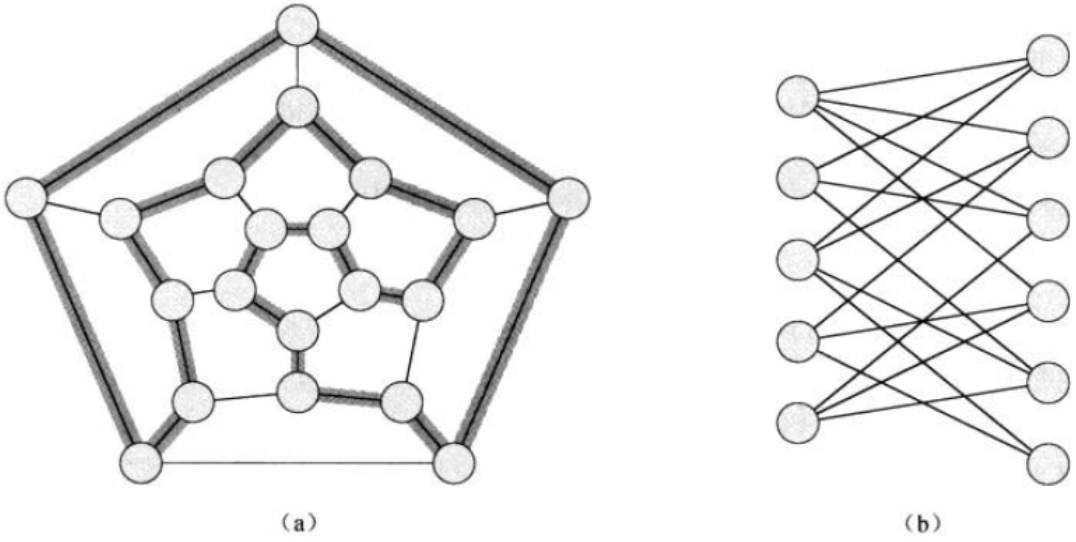



图 34-2 (a)一个表示正十二面体中顶点、边、面的图，其中哈密顿回路以阴影边示出。(b)一个包含奇数个顶点的二分图。任何这样的图都是非哈密顿图

# 典型的NPC问题 (续)

■旅行商问题TSP (Traveling Salesman Problem)

- 一个售货员必须访问$n$个城市，售货员希望恰好访问每个城市一次，并最终回到出发城市。售货员从城市$i$到城市$j$的旅行费用为一个整数$c(i, j)$，旅行所需的全部费用是他旅行经过的各边费用之和，售货员希望整个旅行费用最低。判定问题为：旅行费用不超过$k$。
- 给定一个无向完全图$G=(V, E)$及定义在$V\times V$上的一个费用函数$c$和一个整数$k$，判定$G$是否存在经过$V$中各顶点恰好一次的回路，使得该回路的费用不超过$k$

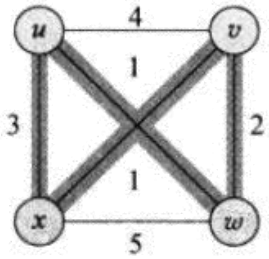



图 34-18 旅行商问题的一个实例。阴影覆盖的边表示费用最低的旅行路线，其费用为 7

# 本章小结

- 算法定义、算法 vs. 程序
- 问题求解步骤
- 算法描述（伪代码）
- 插入排序分析：正确性分析、算法执行时间（最好、最坏、平均情况）
- 渐近表示法（$\Theta/O/\Omega/o/\omega$记号）及函数比较
- 标准记号及常用函数（单调性、取整、模运算、多项式、指数、对数、阶乘、多重函数、多重对数函数、斐波那契数）
- NP完全性理论（P/NP/NPC/NP-Hard，典型的NPC问题）

# 第2章 分治策略

苏州大学 计算机科学与技术学院
汪笑宇
Email: xywang21@suda.edu.cn

# 引入

■凡治众如治寡，分数是也。

——《孙子兵法》

释义：

- 治众：治理人数众多的军队
- 治：治理；分数：军队的组织编制

# 本章内容

- 分治策略（教材Chapter 4）
- 快速排序（教材Chapter 7）
- 中位数和顺序统计量（教材Chapter 9）

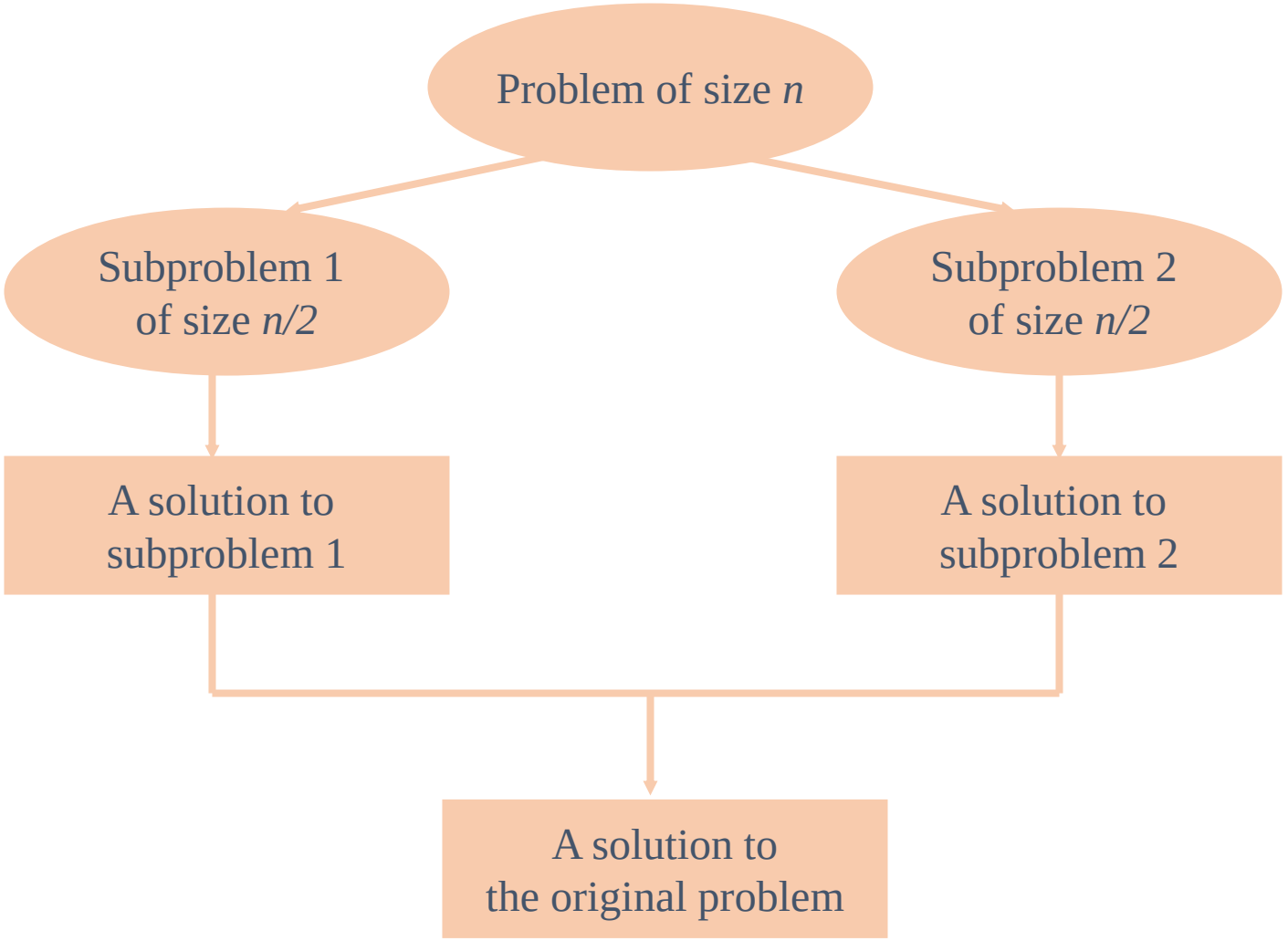
# 主要内容

- 理解递归（Recursion）的概念
- 掌握设计有效的分治策略算法及时间性能分析
- 通过范例学习分治策略设计技巧

# 递归式与分治法

- 直接或间接地调用自身的算法称为递归算法；用函数自身给出定义的函数称为递归函数
- 分治法产生的子问题往往是原问题的较小模式
- 分治与递归像一对孪生兄弟，经常同时应用在算法设计之中，并由此产生许多高效算法

# 递归式与分治法 (续)


Problem of size *n*
Subproblem 1 of size *n/2*
Subproblem 2 of size *n/2*
A solution to subproblem 1
A solution to subproblem 2
A solution to the original problem

# 递归式

■例1：阶乘函数定义

$$n! = \begin{cases} 1, & n = 0 \\ n \cdot (n-1)!, & n > 0 \end{cases}$$

边界条件

递归方程

➢边界条件与递归方程是递归函数的两个要素，递归函数只有具备了这两个要素，才能在有限次计算后得出结果

# 递归式 (续)

■例2：斐波那契数列

$$F(n) = \begin{cases} 0, & n = 0, \\ 1, & n = 1, \\ F(n-1) + F(n-2), & n \geq 2. \end{cases}$$

➢产生的序列为：0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...

➢第$n$个Fibonacci数可递归地计算如下：

```
F(n)
1  if n = 0 then return 0
2  elseif n = 1 then return 1
3  else return (F(n-1) + F(n-2))
```

渐近上界？$O(\phi^n)$

$\phi = \frac{1+\sqrt{5}}{2} \approx 1.618$

空间复杂度？$O(n)$

# 递归式 (续)

■例2：斐波那契数列

➢除了直接递归以外的另3种求解方案：

- 方法1：递推方法 时间$O(n)$
- 方法2：求解通项公式 时间$O(1)$
- 方法3：分治策略 时间$O(\lg n)$

# 递归式 (续)

■例1, 2中的函数都可以找到非递归方式定义：

$$n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n = \prod_{i=1}^{n} i$$

$$F(n) = \frac{\phi^n - \hat{\phi}^n}{\sqrt{5}} = \left\lfloor \frac{\phi^n}{\sqrt{5}} + \frac{1}{2} \right\rfloor$$

$$\phi = \frac{1+\sqrt{5}}{2} \approx 1.618$$

$$\hat{\phi} = \frac{1-\sqrt{5}}{2} \approx -0.618$$

# 递归式 (续)

■例3：Ackermann函数

当一个函数及它的一个变量是由函数自身定义时，称这个函数是双递归函数

Ackermann函数$A(m, n)$定义如下（双变量函数）：

$$A(m,n) = \begin{cases} n+1, & m = 0 \text{ and } n \geq 0, \\ A(m-1,1), & m \geq 1 \text{ and } n = 0, \\ A(m-1, A(m,n-1)), & m \geq 1 \text{ and } n \geq 1. \end{cases}$$

➢$A(m, n)$对$m$的每一个值都定义了一个单变量函数

# 递归式 (续)

■例3：Ackermann函数

$$A(1,n) = 2 + (n+3) - 3$$
$$A(2,n) = 2 * (n+3) - 3$$
$$A(3,n) = 2^{n+3} - 3$$
$$A(4,n) = \underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}_{n+3} - 3$$

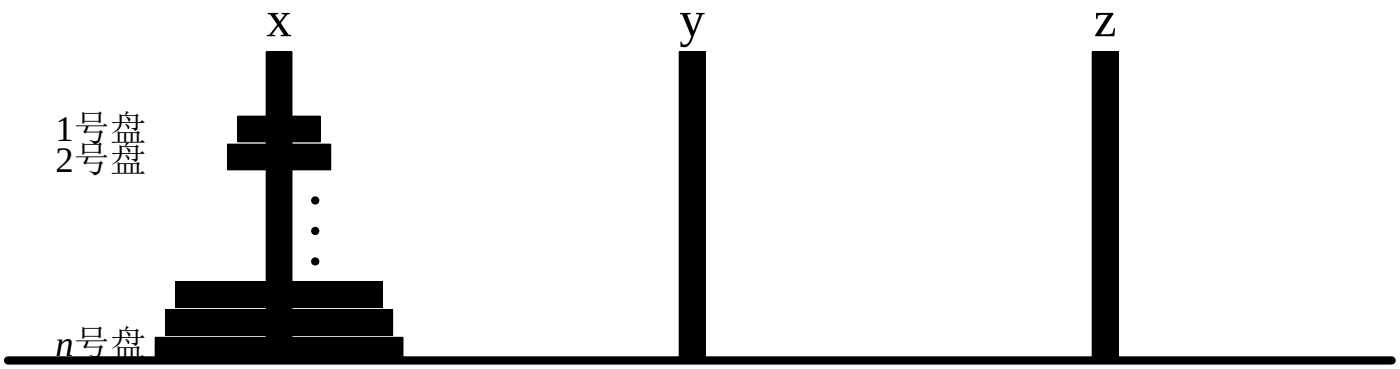
$$A(m,n) = \begin{cases} n+1, & m = 0 \text{ and } n \geq 0, \\ A(m-1,1), & m \geq 1 \text{ and } n = 0, \\ A(m-1, A(m,n-1)), & m \geq 1 \text{ and } n \geq 1. \end{cases}$$

# 递归式 (续)

■例4：*n*阶Hanoi塔问题

将x上的圆盘移到z上，要求按同样次序排列，且满足：

- 每次只能移动一片
- 圆盘可插在x, y, z任一塔座上
- 任一时刻大盘不能压在小盘上



# 递归式 (续)

■例4：*n*阶Hanoi塔问题 Hanoi(*n*, x, y, z)

- 当*n*=0时，没盘子可供移动，什么也不做
- 当*n*=1时，可直接将1号盘子从x移动到z上
- 当*n*=2时，可先将1号盘子移动到y，再将2号盘子移动到z，最后将1号盘子移动到z
- 对于*n*>0的一般情况可采用如下分治策略进行移动：
  1. 将1至*n*-1号盘从x移动至y，递归求解Hanoi(*n*-1, x, z, y)
  2. 将*n*号盘从x移动至z
  3. 将1至*n*-1号盘从y移动至z，递归求解Hanoi(*n*-1, y, x, z)

# 递归式 (续)

■例4：*n*阶Hanoi塔问题

➢分解：设*n*>1，将*n*个圆盘从x移到z，y为辅助塔座：

1. 将上面*n*-1个盘从x移至y，z为辅助塔座
2. 将第*n*号圆盘（最大的圆盘）从x移至z
3. 将y上*n*-1个圆盘移至z，x为辅助塔座

子问题特征属性与原问题相同规模小1，参数不同

➢终结条件：*n*=1时，直接将编号为1的盘子从x移到z

# 递归式 (续)

■例4：*n*阶Hanoi塔问题

```
Hanoi(n, x, y, z)
1  if n = 0 then return
2  elseif n = 1 then move(1, x, z)  //将1号盘子从x移到z
3  else
4     Hanoi(n-1, x, z, y)  //源x、目的y、辅助z
5     move(n, x, z)
6     Hanoi(n-1, y, x, z)  //源y、目的z、辅助x
```

➢时间复杂度分析：设$T(n)$表示$n$个圆盘的Hanoi塔问题移动圆盘的次数

$$T(n)=2T(n-1)+1$$

- $T(0)=0$
- $n>0$时，第4、6行：$T(n-1)$次；第5行：1次

# 递归式 (续)

■例4：*n*阶Hanoi塔问题

➢时间复杂度分析：设$T(n)$表示$n$个圆盘的Hanoi塔问题移动圆盘的次数，则$T(n)=2T(n-1)+1$

$$
\begin{aligned}
T(n) &= 2T(n-1)+1 \\
&= 2(2T(n-2)+1)+1 \\
&= 2(2(2T(n-3)+1)+1)+1 \\
&\vdots \\
&= 2^n T(n-n)+1+2+\ldots+2^{n-1} \\
&= 2^n-1 \\
&= O(2^n)
\end{aligned}
$$

# 递归式 (续)

■例5：全排列问题
设计递归算法生成$n$个元素$\{r_1, r_2, \ldots, r_n\}$的全排列

- 设$R=\{r_1, r_2, \ldots, r_n\}$是要进行排列的$n$个元素
- 集合$R_i=R-\{r_i\}$，即$R_i$表示$R$去除元素$r_i$的集合
- 集合$X$中元素的全排列记为perm($X$)
- perm($X$)($r_i$)表示在全排列perm($X$)的每一个排列后加上后缀$r_i$得到的排列
- $R$的全排列可归纳定义如下：
  - 当$n$=1时，perm($R$)=$\{(r_1)\}$，其中$r_1$是集合$R$中唯一的元素；
  - 当$n$>1时，perm($R$)={perm($R_n$)($r_n$), perm($R_{n-1}$)($r_{n-1}$), …, perm($R_1$)($r_1$)}

# 递归式 (续)

■例5：全排列问题

```
PERM(A, n)
1  if n = 0 then
2     save(A)  //保存A的全排列
3  else
4     PERM(A, n-1)  //perm(R_n)(A[n])
5     for i ← n-1 downto 1 do
6        exchange A[i] with A[n]  //交换A[i]和A[n]，A[n]作为排除元素
7        PERM(A, n-1)             //相当于原数组A的perm(R_i)(A[i])
8        exchange A[i] with A[n]  //数组A恢复原状
```

■时间复杂度 $T(n) = \begin{cases} T_{\text{save}}, & n = 0, \\ n(T(n-1) + O(1)), & n \geq 1. \end{cases}$ $O(T_{\text{save}} \cdot n!)$